Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 38 (2012) 647 – 654

Procedia Engineering

www.elsevier.com/locate/procedia

International Conference on Modeling, Optimization and Computing (ICMOC 2012)

A Genetic Algorithm Approach for Test case Optimization of Safety Critical Control

Samatha K[a], Shreesha Chokkadi[b], Yogananda Jeppu[c], a*

[a]*Lecturer E&C, SIT, Mangalore, 574143, India.*
[b]*Professor of I&CE, MIT Manipal, 576104, India.*

[c]Software Engineer, Moog India Technology centre, Bangalore, India.

Abstract

Safety plays a key role in the safe operation of any safety critical control systems. Safety in such systems depends on the correct operation of the software meant for the safety purpose. Thorough testing of software is required to avoid the catastrophic accidents or to minimize the failure. As a case study, benchmark problem is tested against the C code according to the required specification of the system. Due to complexity involved in the control system there is a need to create a set of test inputs automatically. This paper describes the generation of optimized test cases to ensure block coverage metrics using Genetic Algorithm and results are compared with the Taguchi design of experiments. Random error seeding is carried out into the code to study the efficacy of the test cases.

Key Words – Aerospace Safety, Automatic Testing, Genetic algorithm, Optimization.

1. Introduction

In recent times an increasing amount of reliance is placed on computers in aircrafts which are used in safety critical systems to add functionality, increase flexibility, controllability and performance. In such cases safety is dependent on the correct operation of the systems. Well known example is aircraft fly by wire control systems, where the pilot inputs commands to the control computer using a joystick, and the computer manipulates actual aircraft controls. The number of such systems adds additional risk on software. Failures of safety critical systems would result in catastrophic loss of life, significant property damage or damage to the environment [1].

Some of recent accidents cited in literature are the following. On April 30, 1999, Titan IV B-32/Centaur TC-14/Milstar-3 failed due to incorrect roll rate filter constant zeroed the roll rate data, resulting in a loss of roll axis control followed by yaw and pitch control [2]. On April 19, 2006 Hartsfield-Jackson Atlanta International airport was shut down because of identification of image of suspicious device and software failed to indicate it as part of its routine testing [3]. For all major disasters, the accident investigation board concluded that the failure was directly or indirectly due to inadequate software development, testing and quality assurance process. This paper emphasizes on proper coding of the software and automated testing of the safety critical control systems especially the flight control systems.

* Corresponding author. Tel.: +91-9480145549;
E-mail address: samatharaok@rediffmail.com.

1877-7058 © 2012 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
doi:10.1016/j.proeng.2012.06.080

Safety systems are realized in such a way, that in case of a failure-free operation it must perform exactly as the specified functionality and in case of occurrence of a failure it must either remain in a safe state or proceed to predefined safe state [4]. Critical control systems can be improved by the integrated process for control law design, coding, and testing [5]. For modern aircraft design, control law is developed by model based design approach using Matlab/Simulink tool. A systematic and correct coding process to implement the given functionality of the model and adequate testing is necessary to avoid the accidents. Correctness of the system is checked by testing. Software in loop integration level of model based testing is considered in this paper.

For designing high quality code of the software for control systems, it is essential to develop test patterns or a set of test cases. The test cases are subset of input space and should be able to identify any fault in the system under test [6]. Using these test cases on the final system or subsystem or software, correct behaviour of the system or software can be verified. Thus the goal of selecting test cases is to execute specific spots in software entity or to satisfy the design specification of system under test [7]. The coverage analysis of the hand written source code i.e. code coverage and the block coverage of the model provides a useful view of test execution and can indicate problems in the code or limitations in the test suite.

Several tools are available or have been developed in house by the Verification and Validation groups to generate test cases automatically. These tools use coverage criteria for auto test generation from Simulink blocks of control systems tool box. Basic principle in all the tools is to generate a random vector and verify code coverage as binary covered or uncovered metric. This paper is about the generation of optimized test cases to ensure block coverage metrics using Genetic Algorithm and results are compared with the Taguchi design of experiments method. A set of mutants are introduced into the code to study the efficacy of all the methods.

The benchmark problem present in [8] is taken as a case study for testing and test case optimization. The benchmark problem consists of control systems blocks which are the ones typically used in a flight control systems or an automobile systems. It has a combination of linear filters, integrators, non-linear blocks like rate limiters and lookup table. There is a combination of logic and time dependency in terms of persistence blocks. This problem has 10 inputs and 7 outputs.

2. Safety Critical Control Systems Testing

Testing aims at showing that the intended and actual behaviours of a system differ, or at gaining confidence that they do not. Testing is done to detect the failure i.e. observable difference between the behaviours of implementation and what is expected on the basis of specification [9]. Failures to detect errors can result in significant financial loss or disaster in the case of safety critical control systems. A safety critical control system testing has three main purposes: validation, verification, and defect finding. Verification is carried out by second level of integration testing i.e. software in loop simulation.

The aim of this is to confirm the code or software. It is a confirmation testing, where one is not interested in what the models do; but interested in determining whether the code developed is having the same behaviour as that of the model. In this level of simulation it is assumed that the model is correct and it is intended to show that for all tests that run, the model and developed code provides the same result.

The safety control blocks are coded in C. For the same input both model and code is compared. This simulation is executed on the host PC. Thus it forms the software in loop simulation.

Safety system considered is in Simulink environment. A function to call C code in the Matlab environment is needed. Matlab executable (MEX) files does this i.e., to call C subroutines from the Matlab as if they were built in function. An embedded Matlab function block helps in composing a Matlab function within a Simulink model. This blocks work with a subset of Matlab language called the embedded Matlab subset, which provides optimizations for generating efficient, production quality C code for embedded application [10]. A “MEX file” refers to source file. It consists of

1. A gate way routine that interfaces C and Matlab data. It is the entry point to the MEX-file shared library.
2. A computational routine written in C that performs the computations we want to implement.

The algorithm for the model is written in C according to the functionality of the model. The C code is written as three separate files: header file, function file and main file. Using MEX, Software In Loop Simulation are verified by running the model and code more than 1000 times with the random selection of amplitude, frequency and bias values for all the 10 inputs by giving sinusoidal signal. For the entire test runs simulation result shows that there is a perfect match between the model and the code.

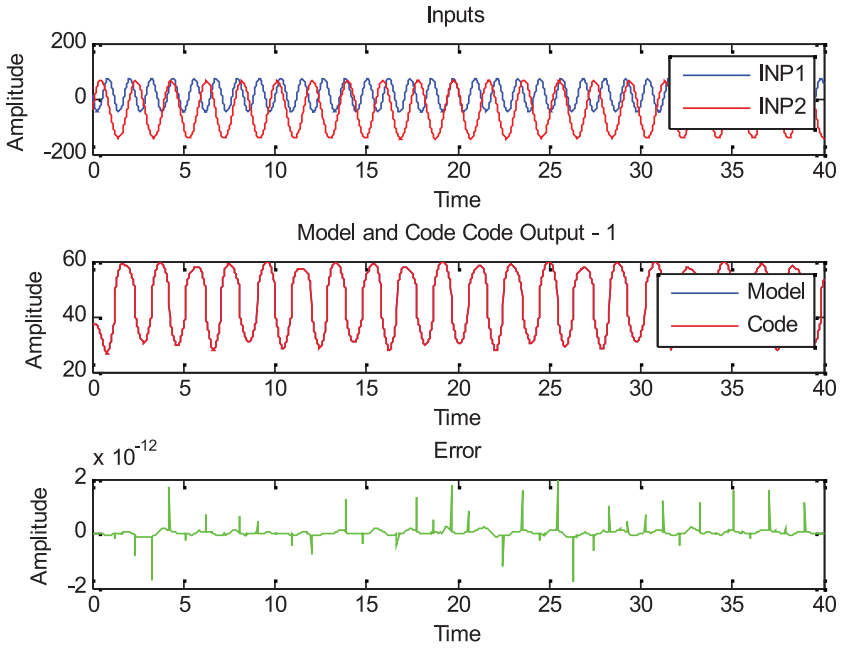

Fig. 1 Software in Loop Simulation Result of First Output of Benchmark Problem

Simulation result is shown only for the first output in the Fig. 1. All the other outputs are matched with the model in the benchmark problem. Any error between the model and code output beyond 10^{-4} is accepted in testing activity.

3. Coverage Metrics

Coverage refers to the extent to which a given verification activity has satisfied its objectives. Coverage Metrics provide a quantitative assessment of a design. Measurement of functionality and structural coverage is a means of assessing the thoroughness of testing. Coverage analyses are used as an aid to generate the test cases by capturing the test deficiencies and new test cases are developed using that information.

There are two measures of coverage. Block coverage metrics define the characteristics of the specific block under test based only on its functional requirements. This metric is independent of the Matlab/Simulink environment. It is defined by considering two pair of cells viz. i) Discrete cell- infers whether the requirements are covered or not. ii) Continuous metric indicates distance to coverage which is to be minimized to ensure coverage [11]. Structural coverage determines how much of the code structure was executed. Structural coverage is necessary to ensure that the requirements based verification process has not missed important features of the implementation.

4. Test Case Optimization

Test cases are generated by considering the functional requirements of the model taken from [11] and are optimized to satisfy block coverage metrics. The efficacy of optimized test cases is checked by introducing mutants into the code.

4.1 Genetic Algorithm (GA) Method

GA is implemented in Matlab to produce the test cases for the benchmark problem. Matlab has a GA toolbox, which plays a major role in producing the test cases. It requires many parameters to be set. The various parameters used are described below.

- Population size: Population size must be at least equal to the number of variables considered. Three parameters are considered as variables for each input. Minimum population size becomes 30. This population size is found to give better result. Thus population size of 30 is considered for the problem.

- Initial Population Range: For generating initial population, all variables are bounded between two limits using this parameter. This is set by the proper understanding of the problem. Different ranges are used for each variable.

- Elite count: is set to 2 so that two best individuals are survived to the next generation without any change.

- Generations: Maximum number of generation GA may take to produce a solution. 10 generations are used. More population size and generation slows down the GA.

- Stopping Criteria considered is the 'generation' to stop the algorithm. For this case 20 generation limit is used. This indicates that at the end of 20 generations, cumulative change in the fitness function value becomes less than 10^{-6} and the algorithm stops resulting in a solution.

- Plot functions: Two plots are considered for the analysis-i) best fittest plot ii) generation stopping criteria to indicate percentage of the stopping criteria met to produce a good solution.

After setting these parameters, Genetic Algorithm is used to produce test cases by using a suitable fitness function to achieve block coverage criteria. This paper deals with the minimization of the fitness function. Fitness function is sum of the product of continuous and discrete metrics. Discrete metrics checks whether the functionality of the blocks are covered or not. This discrete metric has a value at least one to indicate coverage i.e. it has covered. It has checked with less than 1, to identify the discrete metric which are not covered and then it is multiplied with the continuous metric.

Total 51 coverage metrics are considered for the problem from the integrator, rate limier, 1-D look up, delay on-off, and relational block. Each metrics represents a cell in a one dimensional row matrix array. Using the GA set up, GA is executed. In the first run 28, 42, 50 cells of the discrete array are not covered. During the second run fitness function is changed to consider only the effect of those cells which are not covered which results in coverage of the 50th cells. Some other cells are not covered during this run. But it is not needed since these are already covered in the first run. In the third run, fitness function is again changed to cover the uncovered cells which results in a coverage of 28 and 42 cells. Thus total 90 populations with 20 generations cover all the blocks.

GA simulation result showing best fittest plot and percentage of stopping criteria met for single run is shown in the Fig. 2.

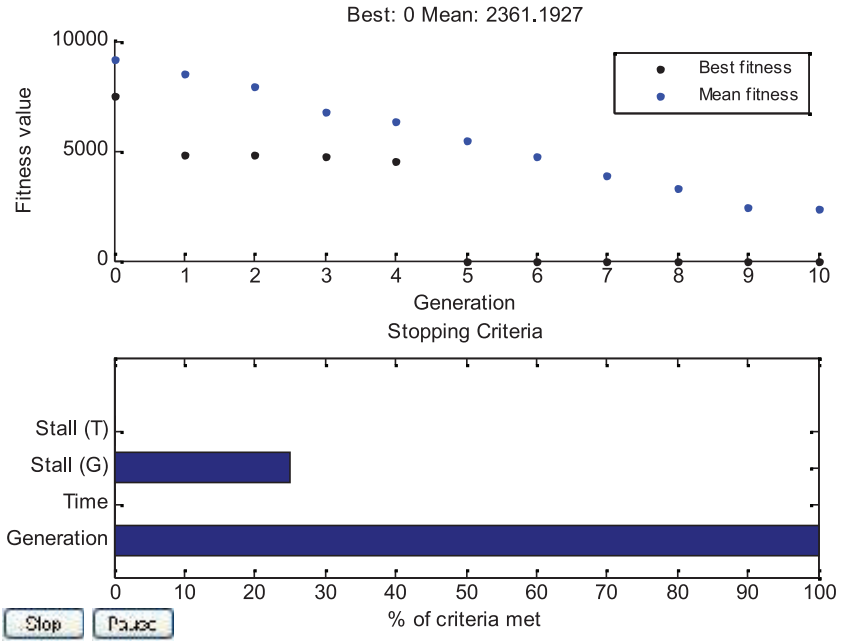

Fig. 2 Genetic Algorithm Plot Showing Best Fitness and % of Criteria Met

Each population of the GA has variables of all the 10 inputs and it forms the test cases. Out of 90 test cases, minimum set of test cases i.e. optimized test cases which are sufficient to cover all the functionality of the block are found using selection algorithm. Selection algorithm is written in Matlab. It takes populations as the input. First, it selects the test case i.e. single population which gives the maximum coverage. This coverage is added with all the other populations considering one population at a time and the one which gives the maximum coverage is retained. The algorithm is repeated till the full coverage is obtained. Four test cases are selected by the selection algorithm.

4.2 Taguchi Method of experiments design

Taguchi method is a technique for designing and performing experiments to investigate processes in which the output depends on many factors without tediously and uneconomically running the process with all possible combinations of values. This method with choosing certain combinations of variables systematically enables to separate individual effects [12]. It uses orthogonal array testing strategy.

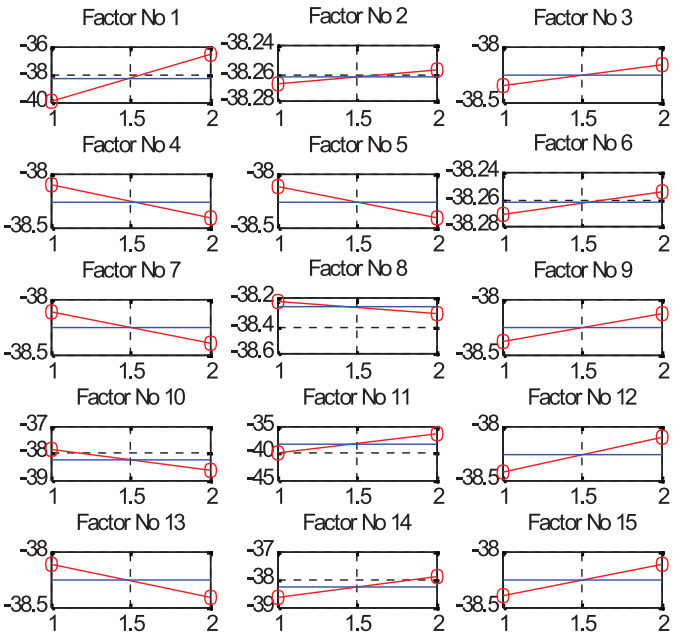

Fig. 3 Showing 15 factors effect – Taguchi Method

In this method, for each level initial value for all the parameters are set with the proper understanding of the functionality of the blocks. For each single run depending on the level value, it will generate different test cases. Same 51 block coverage is considered for this method. Block coverage discrete metrics was observed to identify the number of cells which are not covered. In Taguchi orthogonal array method, it is possible to observe the effect of individual factors to get the 100% coverage. Each factor effect was identified by observing continuous metrics. New test case was generated by analysing continuous metric which has to be covered to get full coverage. If the new test case is unable to show coverage then new variables for the particular input have to be generated. Experiment has to be repeated to achieve complete coverage. Once complete coverage is obtained selection algorithm is used to select the minimum number of test cases which is sufficient to obtain 100% coverage.

The effects on 15 factors are shown in Fig. 3. Remaining 15 factors also produce same kind of the result. X axis indicates the level value. From this plot it is possible to identify which factor level is influencing the performance i.e. to achieve coverage. New test cases are generated based on this. By running different combination of tests depending on the factors effect, 6 test cases cover all the blocks from the selection method.

5. Mutant discussions

Matlab code is written according to the given functionality of the model. The Matlab code was mutated by changing single operator. Four types of mutant set are considered, Arithmetic, logical, relational and data mutant. Examples for arithmetic mutant include changing '+' to '-', '*' etc. in the original script file. Logical mutants include AND to OR, XOR, NAND etc. Relational mutants include '>' to '>=', '<' etc. Data mutants are created by changing values of constants which are used. For example, if we have used UL as 10, it has changed to its near value i.e. 10.2.

All Matlab script file was saved by adding one mutant to the original script file. The optimized test cases selected from the selection algorithm which gives 100% coverage is used to catch these mutants. If any one of the test case identified the mutant then mutant is said to be killed and fault has been identified. Some of the mutants are killed, some are alive and some files are not executed due to initialization errors. Files not executed are ignored.

Summary of the percentage of mutants' killed, alive and not executed files is given in Table 1 and Table 2, by considering all possibilities of four mutants..

TABLE 1. MUTANT ANALYSIS BY GA METHOD

GA Method Mutant Operator	% of Mutants killed	% of Mutants Alive	% of Files which are not executed
Arithmetic	92.9972	7.0028	14.1827
Relation	94.6535	5.3465	12.3264
Data except filter coefficient	84.7826	15.2174	18.2401
Data filter coefficients	85.3082	14.6918	17.7928
Logical	85.473	14.527	18.3899

TABLE 2. MUTANT ANALYSIS BY TAGUCHI METHOD

Taguchi Method Mutant Operator	% of Mutants killed	% of Mutants Alive	% of Files which are not executed
Arithmetic	93.5574	6.4426	14.1827
Relation	95.0495	4.9505	12.3264
Data except filter coefficient	85.1033	14.8967	18.2401
Data filter coefficients	85.6164	14.3836	17.7928
Logical	85.777	14.223	18.3899

TABLE 3. DISCUSSIONS OF THE MUTANTS WHICH ARE ALIVE

Some of the Mutants not killed by both the methods	
Logical operator AND is changed to OR before Output4	Input is masked by the OR and this is not observable
Arithmetic operator + to – is changed in the lower limit of the saturation block after the Input 5.	Both the limits have the same value. It changes the behaviour of the actual saturation block. It is difficult to identify.
Relational operator is changed from > to < after the input1 in saturation block.	This is difficult mutant to catch. Here output is not limited to upper limit since input is not verified between the two bounds > and <.
Denominator first coefficient of the first filter is changed from + to +1+ in the exponential term	This is a very minute error and is difficult to catch
Numerator first coefficient of the fourth filter is changed from 3.4628 to 6.925	It affected behaviour of the filter and it is not identified.
Upper limit of the saturation block after the input 3 is changed from 10 to 9.8	Output of the block is saturated at the value 9.8 when input crosses that value. This type of fault is difficult identify.

6. CONCLUSION

Benchmark problem contains interconnections of basic set of safety critical control blocks which are used in a fly-by-wire controller. This problem is coded in C and SIL simulation is carried out and is verified by randomly selecting its inputs parameters. Automated testing is done for this problem using genetic algorithm and the results are compared with the taguchi design of experiment method. Test cases are optimized by using selection algorithm. Efficacy of test cases is found by introducing four set of errors into the Matlab code. From this it is found that genetic algorithm and Taguchi design of experiment methods are equally good in performance.

This work can further be extended to give special attention for killing maximum percentage of mutants by improving selection of genetic algorithm parameters or by combining two automatic generation techniques.

ACKNOWLEDGEMENT

We would like to thank Nagaraj Murthy, Associate Director, and David Ranson, Managing Director, Moog India Technology Center for giving us this opportunity to work on the problem at MITC as part of our M. Tech coursework. We owe our sincere gratitude to NMAM Institute of Technology, Nitte for extending support during this research work.

REFERENCES

1. B Beizer, “Software Testing Techniques”, 2nd Edition, New York: Van Nostrand Reinhold, 1990
2. John C Knight, “Safety Critical Systems: Challenges and Directions”, Proceedings of the 24th International Conference on Software Engineering Orlando, Florid, 2002.
3. Chetan C U et al. “A New Input-Output Based Model Coverage Paradigm for Control Blocks”, Aerospace Conference, IEEE, 2011.
4. Divyesh Divakar, Samatha K et al. “Optimization of Test case and Coverage Analysis for Model Based Design”, 34th National systems conference on System Solutions for Global Challenges: Energy, Environment and Security, 10-12 December 2010.
5. K M Wanie, H Neir, H Schmid, “Integrated Flight Control Law Design onboard Code Generation and Testing for Safety Critical Applications”, European Conference for Aerospace Sciences, 2005.
6. K Rastocny, “Risk Analysis of Safety Critical Control Systems”, Advances in Electrical and Electronic Engineering, University of Žilina, Univerzitná 8215/1, SK-010 26 Žilina, Slovakia.

7. Mark Utting, Alexander Pretschner and Bruno Legeard, “A Taxonomy of Model Based Testing”, Working Paper Series ISSN 1170-487X, April 2006, pp2.
8. Matlab, Mathworks Central Library. [Courtesy of www.mathworks.com/matlabcentral/ ileexchange/28952-a-benchmark-problem-for-modelbased- control-system-tests-001]
9. Nancy G Leveson, “The Role of Software in Spacecraft Accidents”, AIAA Journal of Spacecraft and Rockets, Vol. 41, No 4, July 2004, pp564-575.Nirmal Kumar Gupta, Mukesh Kumar Rohil, “Using Genetic Algorithm for Unit Testing of Object Oriented Software”, IJSSST, Vol. 10, No. 3.
10. R.H. Lochner, J.E. Matar, Design for quality–An introduction to the best of Taguchi and Western methods of statistical experimental design, New York, 1990
11. W Eric Wong et al. “Recent catastrophic Accidents: Investing How Software Was Responsible”, Fourth IEEE International Conference on Secure Software Integration and Reliability Improvement, 2010, pp15-16.